1993

Modeling Flexibility in Distributed Transactions

Ahmed Elmagarmid
Purdue University, ake@cs.purdue.edu

Aidong Zhang

Report Number:

93-060

MODELING FLEXIBILITY IN DISTRIBUTED TRANSACTIONS

Ahmed K. Elmagarmid
Aidong Zhang

CSD-TR-93-060
September 1993

Modeling Flexibility in Distributed Transactions *

Ahmed K. Elmagarmid and Aidong Zhang
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 USA

Abstract

Flexible transactions have been proposed as a means to specify the flexible control flow necessitated by advanced database applications. This paper presents a formal flexible transaction scheme which precisely models flexible control flow in distributed transactions. The criteria required to ensure the consistency and reliability of flexible transactions are also precisely defined.

Keywords: Databases, distributed transactions, flexible transactions

1 Introduction

Experience has indicated that the traditional transaction concept has limited applicability in advanced database applications, such as CAD/CAM, office automation, and software development environments, which usually involve open-ended, long-lived, and cooperative activities. The ACID properties[1] that characterize traditional transactions have been shown to be insufficient to the requirement of advanced database applications. In an attempt to

*Zhang is supported by a Purdue Research Foundation Fellowship and Elmagarmid is supported by the NSF under grant IRI-8857952.

[1]Atomicity, consistency, isolation, and durability [7].

broaden the traditional model, a variety of extended transaction models have been proposed [3].

Among the recent developments in the specification of advanced database applications in a distributed database environment has been the modeling of the flexible control flow of applications. Such flexibility arises through the existence of alternative threads to the completion of a task. In contrast to the traditional distributed transaction model [1], the fundamental characteristic of a flexible distributed transaction model is its provision of alternative choices for the execution of the subtransactions of distributed transactions. As a result, the execution of distributed transactions becomes more resilient to failures, in that the entire transaction may still successfully execute even if individual subtransactions abort. In [11], one such concept, called ConTract, is proposed, in which a set of predefined actions (called steps) is accompanied by an explicit specification (script) of their control flow. Steps are the elementary units of work, while scripts are invariant predicates that explicitly and separately specify control flow relations between the steps of an application. A similar transaction model, termed flexible transactions, has been proposed [4] for use in the multidatabase environment. The definition of flexible transactions takes the form of a high-level applications description. Various applications semantics, such as commit dependencies, abort dependencies, and the acceptable set of successful subtransactions, are captured in the flexible transaction definition [8].

Following their models, it is indicated in both approaches that the properties of atomicity and isolation can be relaxed. However, a precise characterization of the allowable degree of such relaxation has not been provided. It is thus unclear what criteria should be used to ensure that the execution of a flexible distributed transaction forms a unit of consistent and reliable computation. Atomicity and isolation cannot be totally relaxed; they must be maintained to some degree to preserve the consistency and reliability of flexible distributed transactions. A precise characterization of the relaxation of atomicity and isolation is thus an essential predicate to a discussion of flexible distributed transaction management.

In this paper, we formalize a fundamental model for flexible distributed transactions. As with other extended transaction models, such as nested transactions [10], the syntactic struc-

ture of flexible distributed transactions is depicted without the involvement of semantics. By imposing a pair of control flow relations upon subtransactions, we define each flexible distributed transaction as a set of subtransactions upon which a set of alternative $\prec$-partial orders is specified. Each $\prec$-partial order of subtransactions defines a possible execution of the flexible distributed transaction. Based upon this model, we define the criteria required to ensure the consistency and reliability of of the execution of flexible distributed transactions. These criteria provide a foundation for the investigation of the principles of flexible distributed transaction management.

2 The Model

An elementary transaction is, in this paper, a sequence of read and write accessing operations followed by either a commit or an abort termination operation. In a distributed database, the data items are partitioned into a set of local databases at different local sites. A traditional distributed or global transaction is defined as a set of subtransactions where each subtransaction is a transaction accessing the data items at a single local site. The commitment of a distributed transaction is indicated by the commitment of all its subtransactions.

The flexible distributed transaction model supports flexible control flow by specifying two types of dependencies among the subtransactions of a distributed transaction: (1) ordering dependency between two subtransactions; and (2) alternative dependency between two subsets of subtransactions. Below, we shall formally delineate the flexible control flow in the distributed transaction model.

Let $\mathcal{T} = \{t_1, t_2, ..., t_n\}$ be a repertoire of subtransactions and $\mathcal{P}(\mathcal{T})$ the collection of all subsets of $\mathcal{T}$. Let $t_i, t_j \in \mathcal{T}$ and $T_i, T_j \in \mathcal{P}(\mathcal{T})$. We assume two types of control flow relations to be defined on the subsets of $\mathcal{T}$ and on $\mathcal{P}(\mathcal{T})$, respectively: (1) (**precedence**) $t_i \prec t_j$ if t_i precedes t_j ($i \neq j$); and (2) (**preference**) $T_i \rhd T_j$ if T_i is preferred to T_j ($i \neq j$). If $T_i \rhd T_j$, we also say that T_j is an alternative of (or contingent on) T_i and any subtransaction in T_i is *exclusive* of any subtransaction in T_j. Both precedence and preference relations are of an irreflexive transitive nature. In other words, for each $t_i \in \mathcal{T}$, $\neg(t_i \prec t_i)$; and for each

$T_i \in \mathcal{P}(\mathcal{T})$, $\neg(T_i \rhd T_i)$. If $t_i \prec t_j$ and $t_j \prec t_k$, then $t_i \prec t_k$; if $T_i \rhd T_j$ and $T_j \rhd T_k$, then $T_i \rhd T_k$. In the latter case, any subtransaction in one of the three sets is exclusive of any subtransaction in the other two sets.

Thus, the precedence relation defines the correct parallel and sequential ordering dependencies among the subtransactions, while the preference relation defines the priority dependencies among alternative sets of subtransactions for selection in completing the execution of $\mathcal{T}$.

A flexible transaction can be defined as follows:

Definition 1 (Flexible transaction) *A flexible transaction $\mathcal{T}$ is a set of related subtransactions on which the relations of precedence and preference are defined.*

As this basic definition of flexible transactions provides only a vague picture of the structure of flexible transactions, we shall now seek a more precise delineation. Let T_i be a subset of $\mathcal{T}$, with $\prec$ being an irreflexive transitive precedence relation on T_i. We then say that $p_i = (T_i, \prec)$ is a partial order of subtransactions. $(T_i, \prec)$ is a *singular partial order* if there are no $t_1, t_2 \in T_i$ that are exclusive. A singular partial order $(T_i, \prec)$ is *maximal*, abbreviated as $\prec$-partial order, if there is no element in $\mathcal{T}$ which can be added to T_i without destroying the singular nature of $(T_i, \prec)$. The importance of the concept of $\prec$-partial order is that the execution of a flexible transaction is identical to the execution of the subtransactions in one of its $\prec$-partial orders.

Note that two $\prec$-partial orders $(T_i, \prec)$ and $(T_j, \prec)$ of a flexible transaction may not be disjoint; that is, $T_i \cap T_j \neq \emptyset$. Thus, a subtransaction may involve more than one $\prec$-partial order. Let $p_1 = (T_1, \prec)$ and $p_2 = (T_2, \prec)$ be two $\prec$-partial orders of flexible transaction $\mathcal{T}$. We say that p_1 has higher priority than p_2 in $\mathcal{T}$, denoted $p_1 \rightarrow p_2$, if there are $T_{1i} \subseteq T_1$ and $T_{2j} \subseteq T_2$ such that $T_{1i} \rhd T_{2j}$. The execution of a flexible transaction $\mathcal{T}$ at any moment must be uniquely determined. We define a $\prec$-partial order $(T, \prec)$ as *principal* within $\mathcal{T}$ if it has the highest priority to be chosen for completing the execution of $\mathcal{T}$. In other words, there are no $T' \subset \mathcal{T}$ and $T_i \subseteq T$ such that $T' \rhd T_i$. Clearly, for each flexible transaction $\mathcal{T}$, there is only one principal $\prec$-partial order. We state that two subsets $T_j, T_k \subset \mathcal{T}$ have the same

priority if there is a $T_i \subset \mathcal{T}$ such that $T_i \rhd T_j$ and $T_i \rhd T_k$, but $\neg(T_j \rhd T_k)$ and $\neg(T_k \rhd T_j)$. Clearly, the execution of $\mathcal{T}$ will be uniquely determined if there is no $T_i \subset \mathcal{T}$ which has two alternatives with the same priority.

A flexible transaction $\mathcal{T}$ can thus basically be considered as a set of alternative $\prec$-partial orders. We say that this set of alternative $\prec$-partial orders is *well-formed* if there is no $T_i \subset \mathcal{T}$ which has two alternatives with the same priority. We say that a flexible transaction is *well-defined* if its set of alternative $\prec$-partial orders is well-formed. Following these concepts, we can clearly depict the structure of well-defined flexible transactions as follows:

> **Structure of a well-defined flexible transaction:** A flexible transaction $\mathcal{T}$ consists of a set of well-formed alternative $\prec$-partial orders $\{(T_i, \prec), i = 1, ..., k\}$ of subtransactions, with $\bigcup_{i=1}^{k} T_i = \mathcal{T}$.

So far, we have specified a flexible transaction syntactically, as a set of well-formed alternative $\prec$-partial orders of subtransactions that is determined by the two relations of precedence and preference. The semantics of precedence and preference relations may differ in various applications. For example, the precedence relation may refer to the execution order or commitment order of subtransactions. For instance, $t_1 \prec t_2$ may imply that t_2 cannot start before t_1 commits or that the commitment of t_1 must precede the commitment of t_2. Nevertheless, both cases can be generally considered, as there is an ordering dependency defined on the two subtransactions. Similarly, the preference relation defines alternative choices and their priority. For instance, $\{t_i\} \rhd \{t_j, t_k\}$ may imply that t_j and t_k must abort when t_i commits or that t_j and t_k cannot start after t_i has committed. In this situation, $\{t_i\}$ is of higher priority than $\{t_j, t_k\}$ to be chosen for execution.

We shall now consider an example which was presented in [4].

Example 1 *Consider a travel agent information system engaged in arranging a travel schedule for a customer. The travel agent can choose from the following subtransactions:*

t_1*: Order a ticket from Northwest Airlines;*

t_2*: Order a ticket from United Airlines, if* t_1 *fails;*

t_3: *Rent a car from Hertz;*

t_4: *Reserve a room with Hilton;*

t_5: *Reserve a room with Sheraton, if t_4 fails;*

t_6: *Reserve a room with Ramada, if t_4 and t_5 fail.*

In this example, t_1 and t_2 are two alternative subtransactions for ordering a ticket. In this case, t_2 will be executed if subtransaction t_1 fails to achieve its objective. Similarly, t_4, t_5, and t_6 are alternative subtransactions for reserving a room.

There are six $\prec$-partial orders that are defined on the subsets of $\{t_1, t_2, t_3, t_4, t_5, t_6\}$:

$p_1 = (\{t_1, t_3, t_4\}, \prec)$, $p_2 = (\{t_1, t_3, t_5\}, \prec)$, $p_3 = (\{t_1, t_3, t_6\}, \prec)$

$p_4 = (\{t_2, t_3, t_4\}, \prec)$, $p_5 = (\{t_2, t_3, t_5\}, \prec)$, $p_6 = (\{t_2, t_3, t_6\}, \prec)$.

where p_1 is the principal $\prec$-partial order. Since $\{t_1\} \rhd \{t_2\}$ and $\{t_4\} \rhd \{t_5\} \rhd \{t_6\}$ are the specification of all preference relationships, this flexible transaction is well-defined. □

Similarly to sagas [6], a specification language must be provided to allow application programmers to inform the system of the beginning and end of each subtransaction, as well as the precedence and preference dependencies among the subtransactions. The discussion of these aspects is beyond the scope of this paper and is not presented here.

3 Properties

As with traditional transactions, the execution of a flexible transaction must be a unit of consistent and reliable computation. Thus, we must provide the means to guarantee the consistency and reliability of the execution of a flexible transaction. Traditionally, the ACID properties have been used as the justification of the consistency and reliability of transactions. While some of these properties are applicable to the execution of flexible transactions, others are not. We formulate below the fundamental properties of flexible transactions that are sufficient to ensure the consistency and reliability of the execution of flexible transactions.

We shall first discuss the weaker concept of atomicity for flexible transactions. Although

the traditional understanding of atomicity may no longer be required for flexible transactions, a certain degree of atomicity must still be ensured to produce correct executions. Following from the structure of each flexible transaction, we can see that only the effects of the subtransactions in one $\prec$-partial order should be made permanent in local databases. The subtransactions which are not in this $\prec$-partial order may also commit, but their effects must be eventually undone. The exact measure of the weaker atomicity of flexible transactions is stated as follows:

Property 1 (Semi-atomicity) *The execution of a flexible transaction is semi-atomic: either all and only the effects of its subtransactions in one $\prec$-partial order or no partial effects of its subtransactions are made permanent in local databases.*

The traditional consistency property is inherited by flexible transactions. Following the traditional approach, a database state is defined as a mapping of every data item to a value of its domain, and the *integrity constraints* on these data items are used to define database consistency. A database state is considered to be *consistent* if it preserves these database integrity constraints. In a distributed database system, there are two types of integrity constraints: local integrity constraints are defined on data items in a single local site, while global integrity constraints are defined on data items in multiple local sites. As defined for traditional distributed transactions, the execution of a flexible transaction as a single unit should map one consistent distributed database state to another. Thus, a flexible transaction must preserve both local and global integrity constraints. However, unlike traditional distributed transactions, the consistency of flexible transactions requires that the execution of each $\prec$-partial order of its subtransactions must map one consistent distributed database state to another. We state the global consistency property of flexible transactions as follows:

Property 2 (Global consistency) *The global consistency of a flexible transaction requires that the execution of each $\prec$-partial order of its subtransactions transfer the distributed database from one consistent state to another.*

To achieve a high degree of concurrency in the execution of flexible transactions, it was proposed in [4] that the results of the compensatable subtransactions of a flexible transaction be released prior to the commitment of the flexible transaction. The concept of compensation [5] plays an important role in flexible transaction management. Each subtransaction of a flexible transaction is viewed as either compensatable or non-compensatable. A subtransaction is *compensatable* if, after it commits, the effects of its execution can be semantically undone by executing a compensating transaction. A partial execution of a $\prec$-partial order can be discarded only when the effects of its committed subtransactions can be undone. In contrast to sagas [6], a flexible transaction does not require that any of its subtransactions independently preserve distributed database consistency. Thus, any intermediate results of a flexible transaction may be globally inconsistent. However, the partial results of a flexible transaction which do preserve global consistency may be seen by other flexible transactions before the flexible transaction commits. For example, the results of compensatable subtransactions which are globally consistent may be revealed to other flexible transactions, as long as their effects can still be undone if compensation becomes necessary. The ConTract approach proposes that invariant predicates be defined on database states, defining the conditions for the release of partial results. We define a more flexible concept of isolation as follows:

Property 3 (Flex-isolation) *The execution of a flexible transaction is flex-isolated: it may reveal to other flexible transactions those partial results of its $\prec$-partial orders that are globally consistent.*

The durability of flexible transactions may be defined as similar to the traditional concept. For completeness, we provide it as follows:

Property 4 (Durability) *The durability of a flexible transaction requires that, despite failures, the results of all its committed subtransactions be made permanent in databases.*

We say that a flexible transaction management scheme is correct if it guarantees that the execution of flexible transactions will satisfy Properties 1-4. As with traditional transactions, the global consistency of flexible transactions is usually ensured by their writers. It is the

duty of the distributed transaction manager to ensure semi-atomicity, flex-isolation, and durability. By ensuring semi-atomicity, flexible transactions become more resilient to failures than do traditional distributed transactions. By ensuring flex-isolation, the execution of flexible transactions is capable of achieving a high degree of concurrency.

4 Effects on the Execution of Distributed Transactions

The formal modeling of flexible transactions provides a foundation for the theoretical investigation of flexible transaction management. The execution of flexible transactions can be adapted to a variety of circumstances, through which semi-atomicity, global consistency, flex-isolation, and durability are preserved.

Let $\mathcal{T} = \{t_1, ..., t_n\}$ be a flexible transaction with a set $P = \{p_1, ..., p_k\}$ of well-formed alternative $\prec$-partial orders. The subtransactions in different $\prec$-partial orders may simultaneously proceed with their executions. Following the definition of semi-atomicity, only the effects of the subtransactions in one $\prec$-partial order should remain permanent in the distributed database. We term such a $\prec$-partial order of $\mathcal{T}$ a *committed $\prec$-partial order* of $\mathcal{T}$, denoted $\mathcal{T}^{p^c}$. Thus, any committed subtransaction of $\mathcal{T}$ which is not in $\mathcal{T}^{p^c}$ must have its compensating transaction committed. The execution structure of $\mathcal{T}$ is thus dynamically determined by the distributed database state and the protocol to be used.

The execution of $\mathcal{T}$ can proceed in the following flexible manner. Any compensatable subtransactions of $\mathcal{T}$ can be executed and committed unilaterally, as long as the defined precedence relation is followed. The effects of such committed subtransactions must be undone in the event that the $\prec$-partial order to which this subtransaction belongs is aborted. In addition, if $\prec$-partial order p_1 in $\mathcal{T}$ has higher priority than $\prec$-partial order p_2 in $\mathcal{T}$, then p_1 should always be considered before p_2 in determining $\mathcal{T}^{p^c}$. Clearly, by allowing alternative choices to be executed simultaneously, the flexible transaction approach greatly improves the reliability of distributed database transaction management systems.

In a traditional distributed database environment with prepare-to-commit states [1] present in local database systems, ensuring the correct execution of $\mathcal{T}$ is relatively straightforward. Each compensatable subtransaction can unilaterally commit, and the commitment of all non-compensatable subtransactions in one $\prec$-partial order of $\mathcal{T}$ can be controlled by the two-phase commit protocol [1]. This particular $\prec$-partial order is determined by the priority of $\prec$-partial orders that is specified by the preference relation.

It has recently become evident that existing database systems must be integrated into a multidatabase to support applications that access multiple databases. An important requirement of multidatabase systems is the preservation of local system autonomy, an aspect which distinguishes multidatabase systems from traditional distributed database systems. Due to local design autonomy, some local database systems may not support a prepare-to-commit state. Thus, the atomic commitment of distributed transactions in multidatabase systems has been recognized as an open and difficult issue [12]. In such a situation, a local database system is entitled to delay or reject the execution of a subtransaction. The commitment of all non-compensatable subtransactions together in one $\prec$-partial order of $\mathcal{T}$ must be ensured without using prepare-to-commit states. It has been shown [13] that such a goal is in general not achievable. Various approaches using forward and backward recovery [2, 9] have been proposed to address the issue. Methods based upon the traditional distributed transaction model have generated only limited results. For instance, an approach proposed in [9] specifies that a distributed transaction can only have a single pivot subtransaction that is neither compensatable nor retriable. A subtransaction is *retriable* if it is guaranteed to commit after a finite number of retrials when executed from any consistent database state. The semantic atomicity [5] of a distributed transaction can then be preserved by requiring that the compensatable subtransactions be committed before the commitment of the pivot subtransaction, which in turn must be committed before the commitment of the retriable subtransactions. When the pivot subtransaction aborts, all committed subtransactions are compensatable; otherwise, all uncommitted subtransactions can be retried until commitment. In contrast, flexible transactions offer the opportunity to move from an aborted subtransaction to an alternative subtransaction. A $\prec$-partial order may have two pivot subtransactions, as long as one of them has an alternative subtransaction. The stipulation regarding a single pivot

subtransaction can thus be relaxed. As a result, the flexible transaction model enhances substantially the scope of distributed transaction management beyond that offered by the traditional distributed transaction model.

In both situations, an efficient mechanism that controls the release of only globally consistent partial results of flexible transactions needs to be explored. A straightforward and also efficient approach is to release only the results of compensatable subtransactions that are globally consistent. A detailed discussion of flexible transaction management, including the handling of concurrency control and commitment, varies with the particular environment. An in-depth discussion of these aspects is beyond the scope of this paper and is not presented here.

5 Concluding Remarks

In this paper, we have advanced a fundamental flexible transaction model for distributed transactions. Our discussion has included the definition of the fundamental transaction model and the properties of flexible transactions. Adherence to these properties offers flexible transactions greater resilience to failures than traditional distributed transactions.

The establishment of a flexible transaction model and of the essential properties of flexible transactions provides a foundation for the design of an advanced flexible transaction programming language and the investigation of the principles of flexible transaction management in the distributed database environment. Flexible transactions hold particular promise for transaction management in the multidatabase environment.

References

[1] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Databases Systems.* Addison-Wesley Publishing Co., 1987.

[2] Y. Breitbart, A. Silberschatz, and G. Thompson. Reliable Transaction Management in a Multidatabase System. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 215–224, May 1990.

[3] A. K. Elmagarmid and et. al. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers, 1992.

[4] A. K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 507–581, Brisbane, Australia, Aug. 1990.

[5] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Trans. Database Syst.*, 8(2):186–213, June 1983.

[6] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the ACM Conference on Management of Data*, pages 249–259, May 1987.

[7] J. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the International Conference on Very Large Data Bases*, pages 144–154, Cannes, France, Sept. 1981.

[8] Y. Leu, A. Elmagarmid, and N. Boudriga. Specification and execution of transactions for advanced database applications. *Information Systems*, 17(2), 1992.

[9] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. A transaction model for multidatabase systems. In *Proceedings of International Conference on Distributed Computing Systems*, June 1992.

[10] J. E. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing.* PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1981.

[11] A. Reuter. Contract: A means for extending control beyond transaction boundaries. In *Proceedings of the 2nd International Workshop on High Performance Transaction Systems*, September 1989.

[12] A. Silberschatz, M. Stonebraker, and J. Ullman. Database systems: Achievements and opportunities. *Communication of ACM*, 34(10):110–120, 1991.

[13] N. Soparkar, H. F. Korth, and A. Siberschatz. Failure-resilient transaction management in multidatabases. *IEEE Computer*, 24(12):28–36, December 1991.